# Common constraint mistakes

SUPPLY CHAIN ANALYTICS IN PYTHON

**Aaren Stubberfield**
Supply Chain Analytics Mgr.

datacamp

# Dependent demand constraint

Context

- Production Plan
- Planning for 2 products *(A, and B)*
- Planning for production over 3 months *(Jan - Mar)*
- Product A is used as an input for production of product B

Constraint Problem

- For each unit of B, we must also have at least 3 units of A

# Dependent demand constraint

For each unit of B, we must also have at least 3 units of A

- $3B \leq A$
- $3(2) \leq A$
- $6 \leq A$

Common Mistake:

- $B \leq 3A$
- $3B = A$

# Code example

```
from pulp import *
demand = {'A':[0,0,0],'B':[8,7,6]}
costs = {'A':[20,17,18],'B':[15,16,15]}

# Initialize Model
model = LpProblem("Aggregate Production Planning",
                  LpMinimize)

# Define Variables
time = [0, 1, 2]
prod = ['A', 'B']
X = LpVariable.dicts(
     "prod", [(p, t) for p in prod for t in time],
      lowBound=0, cat="Integer")
```

```
# Define Objective
model += lpSum([costs[p][t] * X[(p, t)]
                for p in prod for t in time])

# Define Constraint So Production is >= Demand
for p in prod:
    for t in time:
        model += X[(p, t)] >= demand[p][t]
```

# Code example continued

```
for t in time:
    model += 3*X[('B',t)] <= X[('A',t)]
```

# Extended constraint

For each unit of B, we must also have at least 3 units of A and ***account for direct to customer sells of A***.

- $3B + Demand_A \leq A$

# Combination constraint

Context

- Warehouse distribution plan
- 2 warehouses *(WH1, and WH2)*
- We ship 2 products *(A, and B)* from each warehouse
- Warehouse WH1 is small and can either ship 12 A products per a week or 15 B products per a week

Constraint Problem

- What combinations of A, or B can be shipped in 4 weeks?

- 1 week only: (1/12)A + (1/15)B ≤ 1

Correct Form

- (1/12)A + (1/15)B ? ≤
- (1/12)(32) + (1/15)(20) ≤ 4
- (32/12) + (20/15) ≤ 4
- 4 ≤ 4

Common Mistakes

- 12A + 15B ≤ 4
- (1/12)A + (1/15)B = 4

```
from pulp import *
import pandas as pd
demand = pd.read_csv("Warehouse_Constraint_Demand.csv", index_col=['Product'])
costs = pd.read_csv("Warehouse_Constraint_Cost.csv", index_col=['WH','Product'])

# Initialize Model
model = LpProblem("Distribution Planning", LpMinimize)

# Define Variables
wh = ['W1','W2']
prod = ['A', 'B']
cust = ['C1', 'C2', 'C3', 'C4']
X = LpVariable.dicts("ship", [(w, p, c) for c in cust for p in prod for w in wh],
                     lowBound=0, cat="Integer")
```

# Code example continued

```
# Define Objective
model += lpSum([X[(w, p, c)]*costs.loc[(w, p), c]
                for c in cust for p in prod for w in wh])

# Define Constraint So Demand Equals Total Shipments
for c in cust:
    for p in prod:
        model += lpSum([X[(w, p, c)] for w in wh]) == demand.loc[p, c]
```

# Code example continued

Constraint

```
model += ((1/12) * lpSum([X['W1', 'A', c] for c in cust])
          + (1/15) * lpSum([X['W1', 'B', c] for c in cust])) <= 4
```

# Extend constraint

Warehouse WH1 is small and either ship 12 A products per a week, 15 B products per a week, ***or 5 C products per a week***. What combinations of A, B, or C can be shipped in 4 weeks?

- $(1/12)A + (1/15)B + (1/5)C \leq 4$

# Summary

- Common Mistakes
  - Dependent constraint
  - Combination selection constraint
- How to extend constraints
- Check constraint by plugging in a value

# Let's practice

SUPPLY CHAIN ANALYTICS IN PYTHON

# Capacitated plant location - case study P2

SUPPLY CHAIN ANALYTICS IN PYTHON

**Aaren Stubberfield**
SuSupply Chain Analytics Mgr.

datacamp

# Capacitated plant location model

Modeling

- Production at regional facilities
  - Two plant sizes (low / high)
- Exporting production to other regions
- Production facilities open / close

# Decision variables

What we can control:

- $x_{ij}$ = quantity produced at location **_i_** and shipped to **_j_**
- $y_{is}$ = 1 if the plant at location **_i_** of capacity **_s_** is open, 0 if closed
  - s = ***low*** or ***high*** capacity plant

# Constraints

- Total Production = Total Demand
  - $\sum_{i=1}^{n} x_{ij} = D_j$ for $j = 1, ..., m$
  - $n$ = number of production facilities
  - $m$ = number of markets or regional demand points

# Constraints

- Total Production ? Total Production Capacity
  - $\sum_{j=1}^{m} x_{ij}$ ? $\sum_{s=1} K_{is} y_{is}$
  - $K_{is}$ = potential production capacity of plant **_i_** of size **_s_**

```
from pulp import *

# Initialize Class
model = LpProblem("Capacitated Plant Location Model", LpMinimize)

# Define Decision Variables
loc = ['A', 'B', 'C', 'D', 'E']
size = ['Low_Cap','High_Cap']
x = LpVariable.dicts("production_", [(i,j) for i in loc for j in loc],
                     lowBound=0, upBound=None, cat='Continuous')
y = LpVariable.dicts("plant_", [(i,s) for s in size for i in loc], cat='Binary')

# Define Objective Function
model += (lpSum([fix_cost.loc[i,s]*y[(i,s)] for s in size for i in loc])
          + lpSum([var_cost.loc[i,j]*x[(i,j)] for i in loc for j in loc]))
```

# Code example continued

```
# Define the Constraints
for j in loc:
    model += lpSum([x[(i, j)] for i in loc]) == demand.loc[j,'Dmd']
for i in loc:
    model += lpSum([x[(i, j)] for j in loc]) <= lpSum([cap.loc[i,s]*y[(i,s)]
                                                       for s in size])
```

# Summary

Capacitated Plant Location Model:

- Constraints
  - Total Production = Total Demand
  - Total Production ≤ Total Production Capacity

# Review time

SUPPLY CHAIN ANALYTICS IN PYTHON

# Solve the PuLP model

SUPPLY CHAIN ANALYTICS IN PYTHON

**Aaren Stubberfield**
Supply Chain Analytics Mgr.

datacamp

# Common modeling process for PuLP

1. ~~Initialize Model~~
2. ~~Define Decision Variables~~
3. ~~Define the Objective Function~~
4. ~~Define the Constraints~~
5. Solve Model
   - call the `solve()` method
   - check the status of the solution
   - print optimized decision variables
   - print optimized objective function

# Solve model - solve method

```
.solve(solver=None)
```

- `solver` = Optional: the specific solver to be used, defaults to the default solver.

```
# Initialize, Define Decision Vars., Objective Function, and Constraints
from pulp import *
import pandas as pd
model = LpProblem("Minimize Transportation Costs", LpMinimize)
cust = ['A','B','C']
warehouse = ['W1','W2']
demand = {'A': 1500, 'B': 900, 'C': 800}
costs = {('W1','A'): 232, ('W1','B'): 255, ('W1','C'): 264,
         ('W2','A'): 255, ('W2','B'): 233, ('W2','C'): 250}
ship = LpVariable.dicts("s_", [(w,c) for w in warehouse for c in cust],
                        lowBound=0, cat='Integer')
model += lpSum([costs[(w, c)] * ship[(w, c)] for w in warehouse for c in cust])
for c in cust: model += lpSum([ship[(w, c)] for w in warehouse]) == demand[c]

# Solve Model
model.solve()
```

# Solve model - status of the solution

```
LpStatus[model.status]
```

- **Not Solved:** The status prior to solving the problem.
- **Optimal:** An optimal solution has been found.
- **Infeasible:** There are no feasible solutions (e.g. if you set the constraints $x \leq 1$ and $x \geq 2$).
- **Unbounded:** The object function is not bounded, maximizing or minimizing the objective will tend towards infinity (e.g. if the only constraint was $x \geq 3$).
- **Undefined:** The optimal solution may exist but may not have been found.

[1] Keen, Ben Alex. “Linear Programming with Python and PuLP [2] Part 2.” _Ben Alex Keen_, 1 Apr. 2016, benalexkeen.com/linear-programming-with-python-and-pulp-part-2/._{{5}}

```
# Initialize, Define Decision Vars., Objective Function, and Constraints
from pulp import *
import pandas as pd
model = LpProblem("Minimize Transportation Costs", LpMinimize)
cust = ['A','B','C']
warehouse = ['W1','W2']
demand = {'A': 1500, 'B': 900, 'C': 800}
costs = {('W1','A'): 232, ('W1','B'): 255, ('W1','C'): 264,
         ('W2','A'): 255, ('W2','B'): 233, ('W2','C'): 250}
ship = LpVariable.dicts("s_", [(w,c) for w in warehouse for c in cust], lowBound=0, cat='Integer')
model += lpSum([costs[(w, c)] * ship[(w, c)] for w in warehouse for c in cust])
for c in cust: model += lpSum([ship[(w, c)] for w in warehouse]) == demand[c]
# Solve Model
model.solve()
print("Status:", LpStatus[model.status])
```

```
Status: Optimal
```

Print variables to standard output:

```
for v in model.variables():
    print(v.name, "=", v.varValue)
```

Pandas data structure:

```
o = [{A:ship[(w,'A')].varValue, B:ship[(w,'B')].varValue, C:ship[(w,'C')].varValue}
     for w in warehouse]
print(pd.DataFrame(o, index=warehouse))
```

- loop model variables
- store values in a pandas DataFrame

```
# Solve Model
model.solve()
print(pulp.LpStatus[model.status])
o = [{A:ship[(w,'A')].varValue, B:ship[(w,'B')].varValue, C:ship[(w,'C')].varValue}
     for w in warehouse]
print(pd.DataFrame(o, index=warehouse))
```

Output:

```
Status: Optimal
|       |A      |B      |C      |
|:------|:------|:------|:------|
|W1     |1500.0 |0.0    |0.0    |
|W2     |0.0    |900.0  |800.0  |
```

# Solve model - optimized objective function

Print the value of optimized objective function:

```
print("Objective = ", value(model.objective))
```

```
# Solve Model
model.solve()
print(pulp.LpStatus[model.status])
output = []
for w in warehouse: t = [ship[(w,c)].varValue for c in cust] output.append(t)
opd = pd.DataFrame.from_records(output, index=warehouse, columns=cust)
print(opd)
print("Objective = ", value(model.objective))
```

```
Status: Optimal
|       |A      |B      |C      |
|:------|:------|:------|:------|
|W1     |1500.0 |0.0    |0.0    |
|W2     |0.0    |900.0  |800.0  |
Objective = 757700.0
```

# Summary

Solve Model

- Call the `solve()` method
- Check the status of the solution
- Print values of decision variables
- Print value of objective function

# Let's practice!

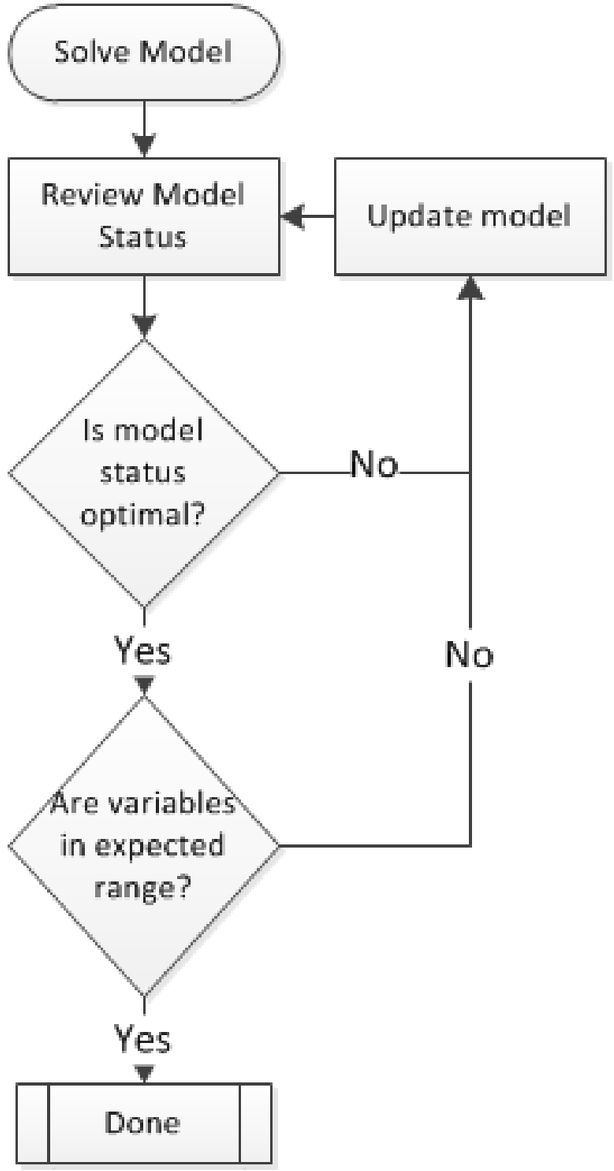
SUPPLY CHAIN ANALYTICS IN PYTHON

# Sanity checking the solution

SUPPLY CHAIN ANALYTICS IN PYTHON

**Aaren Stubberfield**
Supply Chain Analytics Mgr.

datacamp


Solve Model
Review Model Status
Update model
Is model status optimal?
No
Yes
No
Are variables in expected range?
Yes
Done

# Check the model status

- **Infeasible:** There are no feasible solutions.
  - Review the constraints
- **Unbounded:** The object function is not bounded, maximizing or minimizing the objective will tend towards infinity.
  - Review the objective function
- **Undefined:** The optimal solution may exist but may not have been found.
  - Maybe the best available solution
  - Review how you are modeling the problem

# Check if results are within expectations

Are the **decision variables** and value of **objective** within expected range?

- Based on knowledge / understanding of problem
- If "Yes", then you have a valid solution
- If "No", then review:
  - Python code
  - Data
  - Write the LP File

# Write LP

```
writeLP(filename)
```

- `filename` = The name of the file to be created

Shows:

- Name of problem
- Objective function and if minimizing or maximizing
- Constraints, including constraints on Decision Variables called Bounds
- Decision variables

# Code example

```
\* Aggregate Production Planning *\
Minimize
OBJ: 20 prod_('A',_0) + 17 prod_('A',_1)
    + 18 prod_('A',_2) + 15 prod_('B',_0)
    + 16 prod_('B',_1) + 15 prod_('B',_2)
Subject To
_C1: prod_('A',_0) >= 0
_C2: prod_('A',_1) >= 0
_C3: prod_('A',_2) >= 0
_C4: prod_('B',_0) >= 8
_C5: prod_('B',_1) >= 7
_C6: prod_('B',_2) >= 6
```

```
Bounds
0 <= prod_('A',_0)
0 <= prod_('A',_1)
0 <= prod_('A',_2)
0 <= prod_('B',_0)
0 <= prod_('B',_1)
0 <= prod_('B',_2)
Generals
prod_('A',_0)
prod_('A',_1)
prod_('A',_2)
prod_('B',_0)
prod_('B',_1)
prod_('B',_2)
```

# Summary

Strategy for Sanity Checking

- Check the model status
- Check decision variables and objective inside expected range
- Use `writeLP()` if needed

# Practice time!

SUPPLY CHAIN ANALYTICS IN PYTHON